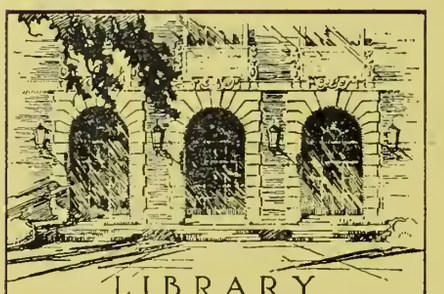


REPORT NO. 196

MATHEMATICS

COO-1469-0013

AN INTERACTIVE, TIME SHARE SYSTEM FOR A SMALL COMPUTER

by

Melvin E. Haas

February 1, 1966

DEPARTMENT OF COMPUTER SCIENCE · UNIVERSITY OF ILLINOIS · URBANA, ILLINOIS

Report No. 196

AN INTERACTIVE, TIME SHARE SYSTEM FOR A SMALL COMPUTER*

by

Melvin E. Haas

February 1, 1966

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

* This work supported in part by Contract No. US AEC AT(11-1)1469, under the direction of Professor C. W. Gear, and was submitted in partial fulfillment for the degree of Master of Science in Aeronautical and Astronautical Engineering, January, 1966.

ACKNOWLEDGMENT

The author would like to express his appreciation for the guidance provided by Professor Harry H. Hilton and Professor Gernot Metze. The technical counsel given by Professor Metze is particularly appreciated.

Thanks are also due Professor C. W. Gear and the Department of Computer Science for supporting this work.

TABLE OF CONTENTS

1. INTRODUCTION

The motivation for an interactive computer system is to obtain fast response for program generation, debugging, and running. The human user retains control of the computing process, and can modify program, change parameters, and correct errors as required during solution of the problem. An interactive system contains a man-computer interface as a distinguishing characteristic. This interface is not present in the more common, batch process, computer systems.

The requirements and characteristics of the human user are the major design factors for an interactive system. The human engineering of the system determines the efficiency of computer use. A system that is overly complicated, irrational, or slow reacting will waste problem solving time. An ideal interactive computer system has the following attributes:

1. short, simple, nonambiguous control language,
2. fast response to user commands,
3. complete human control of computer actions,
4. routine labor supplied by the computer.

An interactive computer system involves real-time communication with a human, at a speed very much slower than the computer speed, allowing one computer to simultaneously handle several users, with the computing of one user overlapped with the communications of the others. Conflicts between users are handled by sharing the computer time, with each user getting small segments of the available time on a rotating basis. Time-sharing allows an interactive system to make economic use of the computer.

Interactive, time share computer systems have been in existence for several years.[1] Most of these require the use of a powerful computer with huge

---

[1] Several of these systems are described in references 1, 3, 4, 9, 11, and 14.

memory capacity. Small computers usually have systems designed to maximize the computational power available, and are rarely interactive or time-sharing. By limiting the computing power offered, an interactive, time share system can be put on a small computer. The work detailed in this paper is the design of such a system.

The motivation for an interactive system for a small computer is to obtain fast response for program generation, debugging, and running. The compromises made to compensate for the limitations of the computer affect only the size of the program and the extent of the computations allowed. The problems of system design, other than human engineering, reduce to choosing features that allow reasonable computer power, with minimum demands on the computer.

For the purpose of this design, a small computer will be defined as having the following minimum capacities:

1. the ability to perform floating point arithmetic by subroutine or hardware,
2. the ability to communicate with as many keyboard/teleprinters as there are to be simultaneous users,
3. a minimum of 75,000 bits of core memory, plus at least 5,000 additional bits for each user.[2]

Additional computer features may be incorporated, but are not essential to the system design. Initial implementation of the system is to be made on a PDP-7 computer,[3] with teletype consoles for up to 5 users. Techniques developed in this report are illustrated by, but not restricted to this one implementation. A more powerful computer may use the same system with more users, and conversely.

---

[2]The numbers given here are only estimates, and are not substantiated.

[3]The PDP-7 is a small, general purpose computer manufactured by the Digital Equipment Corporation, Maynard, Massachusetts. See reference 5.

The design of the computer system is presented in three sections. The first section presents the language, LTLTRAN, by which each user communicates with the computer. The second section develops the time share aspects of the system; the features that keeps each user from adversely affecting the others. The third section details the computer realization of the system, with particular emphasis on methods of organization and planning.

2. THE LANGUAGE

It is not necessary to restrict a time share computer system to one input language. Several such systems[1] allow the user to define or use a number of different computer languages. In multiple language systems, there is very little control over the human-computer interface, but restricting a system to one language[2] allows the system to be optimized for human communication. The disadvantage of the single language system is that the computational power offered to the user is restricted to that allowed by the language. This restriction may be used to advantage in the design of a system for a small computer, since the power of the language may be tailored to fit the capabilities of the computer. Thus a single language system allows the system design to be both human engineered and matched to the computer capabilities. A single language system has the additional advantage of allowing system programs, such as the language processor, to be shared by all users. This represents a significant storage space savings within the small computer.

The computations most suitable for performance on a small computer fall in the area of arithmetic problem solving, rather than in the area of data processing since the limited memory capacity of a small computer does not allow a large amount of data to be handled.

The most widely used arithmetic problem solving computer language is FØRTRAN.[3] LTLTRAN, the language designed for a small computer system is a subset of FØRTRAN. This decision was based on the assumption that the human user would likely be familiar with FØRTRAN, and would therefore make fewer errors and require less training in the use of LTLTRAN. To eliminate a source of user confusion, the features of LTLTRAN are similar to the corresponding

---

[1] Project MAC and SDS/Berkeley systems are examples. See references 3 and 11.

[2] BASIC and QUIKTRAN are examples. See references 4 and 9.

[3] FØRTRAN is described in references 7, 8, and 14.

features in FØRTRAN, wherever possible.

The formal, complete syntax and semantics of LTLTRAN are contained in Appendix I and Appendix II of this report. The remainder of this section presents the design basis of the language.

The major design feature of LTLTRAN is the limit placed on the number of variables allowed. The use of 26 variables, addressed by single letters of the alphabet, sets the limit on the computational power of the language. This addressing scheme is simple for the human user, as it corresponds to typical algebraic usage, and is free of tricky restraints. Use of computer storage is minimized, because no name-to-storage mapping tables need be maintained. The remaining features of LTLTRAN are those needed to allow reasonable use of the 26 variables.

All variables and constants are handled in floating point notation (exponent and fraction) to simplify the programming of both the user's program and the system. This is not a restriction, because fixed point notation (integer only) is a logical subset of floating point notation. The only complication caused by this decision arises in the semantics of LTLTRAN exponentiation, as will be discussed.

The LTLTRAN conventions of arithmetic operations and function use are exactly those of FØRTRAN except for exponentiation. FØRTRAN exponentian conventions are determined by the notation mode of the exponent, as follows:[4]

	argument	exponent	result
A.	fixed point	fixed point	fixed point with correct sign
B.	fixed point	floating point	not allowed
C.	floating point	fixed point	floating point with correct sign
D.	floating point	floating point	floating point absolute value

[4] FØRTRAN conventions are described in reference 8.

LTLTRAN does not distinguish between fixed and floating point exponents and therefore cannot follow FØRTRAN conventions. On the assumption that a common use of LTLTRAN would be to evaluate polynomial expressions, convention C above has been chosen for LTLTRAN. The exponent is assumed to be an integer in a LTLTRAN expression. If at program execution time, an exponent is found to contain a fractional part, an error stop will occur. Convention D above may be programmed in LTLTRAN by use of the ABS, LØG, and EXP functions.

Examples: A = -2 ** 3 would produce A = -8

A = -2 ** 3.2 would produce an error stop

A = EXP (LØG (ABS (-2)) * 3.2) would produce A = 9.18959

The basic unit of the LTLTRAN input is the statement (input line). All program transfers are to the beginning of a statement. To provide for full control by the human user, a double labeling scheme is provided. One label, the line number, is provided by the computer for every statement. The other label, the statement number, may be supplied by the user where desired. Either label may be used in any control statement. To prevent confusion between labels, the statement number is limited to 2 digits, and the 3 digit line number is in the form of a decimal fraction.

LTLTRAN control statements allow any extent of branching and looping. The functions of all FØRTRAN transfer operations may be programmed in LTLTRAN by use of the IF and GØ TØ statements. LTLTRAN subroutines are defined by the label on the first statement in the routine. Transfer to a subroutine by a CALL statement is an extension of the GØ TØ statement actions. The subroutine RETURN statement serves as the depository of the return label, thus retaining consistency with FØRTRAN subroutine conventions. LTLTRAN subroutine calls may be nested to any depth.

The line number label serves to keep the user informed of the actions of the computer. The numerical order of the line numbers indicates the order

of computation (in the absence of control statements). Replacement lines and insert lines are put in proper position as indicated by the line number. Upon program execution of input and output statements, the line number of the statement is typed to help the user identify the program action being executed. At all error stops, the line number of the statement that caused the error is typed, along with a code indicating the error cause.

Each LTLTRAN input statement is checked for format errors immediately. If an error is found, a standard error stop occurs, and the user may then type in a corrected statement. Thus the user may correct this type of error locally, while his thoughts are still on the particular statement. The LTLTRAN convention of one statement per line, and the terseness of the statements minimizes the amount of typing required to correct an error.

At the execution of a READ statement, the computer types the letters of the variables wanted, thus reducing confusion and the chance of error on input. As an additional convenience, a plus sign, +, may be typed instead of an input value to indicate no change in the value of the variable.

The command (immediate execution), mode of LTLTRAN operation (indicated by a preceding number sign, #) allows "desk calculator" type use of the console. This mode of operation may also be used to set values of variables before, or during, the run of a program. A transfer command may be used to initiate execution at any statement. A command PRINT statement may be used to obtain the current value of any variables.

To facilitate user control of computer operation, the user may interrupt program execution at any time by typing any printing character. Conditions are saved at interruption, so that execution may be resumed by use of the GØ ØN command.

Nonprinting console characters are ignored at all times. The user may format the printed page at will, by use of tab, line feed, and space.

Comments may be included, and errors logically erased, by use of the per cent, %, line deletion convention. Single letter errors may be corrected by use of the backspace.

The user has to supply only the minimum input information to specify the actions desired. No unnecessary typing is called for from the user, and output typing from the computer is held to a minimum. The teletype console types at a maximum rate of one character every .1 second. A human reaction rate is of the same order.[5] Any excess typing after the information content has been sensed by the human is a waste of time, and may be very annoying to the user.

---

[5]Human reaction times to light and sound stimulus fall between .050 and .400 seconds typically. (reference 15)

3. SYSTEM OPERATION

The system operation should be transparent to the user. That is, the user should only be concerned with his own LTLTRAN program, and should not be affected by the actions of the system or other users. The purpose of the system control is to present each user with the appearance of an independent version of LTLTRAN. The major design problems in the system control fall in the areas of response time and fail safe operation.

Response time is the time it takes the computer to respond to the user's request for action. To the LTLTRAN user, there are three types of request for computer response:

1. inputting of a character,
2. finishing a line of input,[1]
3. initiating execution of a program.

Each of these requests has a different maximum acceptable response time.

Character input from the consoles requires the shortest response time. It is unthinkable for the system to require the user to type slowly, because the system is not able to read the characters at fast typing speed. To the user, the system must give "instantaneous" service for each character at all times and at all typing speeds. With typical teletype equipment,[2] there is a 0.018 second interval during which a character may be read after it has been received at the computer. This interval (or "window") sets the maximum execution time of any system routine on computers without hardware interrupt. On a computer with interrupt features, the character window sets the maximum time for continuous interrupt disallowed operation. Section 4 will discuss character handling in detail.

---

[1] User intervention during execution is handled in the same manner as an input line.

[2] Teletype handling requirements are detailed in references 6 and 10.

When the user finishes an input line (by typing the end of line character, SO), he expects the computer to decode the line, check for errors, and return with the line heading for the next line. The system control should be designed to give very fast service to input lines because computer caused delays in the input process are most annoying to the human. If the response comes from the computer in less than a human reaction time (.05 sec. to .4 sec.), the user will not be aware of any delay. Response time greater than 2 seconds may distract the user from the problem at hand.

The system control has little effect on the running time of a user's program. The running speed of a program is a function of the type of program and the power of the computer. The execution response time to the LTLTRAN user is a function of the running speed of his program, and the number of computer actions required by other users. With a long program, several other users, and a slow computer, the response time for execution might be on the order of minutes. The major task of the system control in program execution is to assure that each user gets a fair share of the available computing time. The system control should be able to stop execution of a user's program, and later resume execution at the same point. The point of execution stop should always occur at a program point corresponding to the transition between input LTLTRAN statements. If the system never stops during the execution of a LTLTRAN statement, no changes that the user makes to the program can cause a system failure. This point will be discussed later under the topic fail safe operation.

The execution stopping process includes: recognizing the transition point between LTLTRAN input statements; saving all the program variables required to restart; and transferring to the next action to be performed. Upon receipt of an end of line character from a console, the program in execution must be stopped and saved, and the input line decoded.

The response time for processing an input statement is the total computer time required to: recognize the end of line character, finish execution of the program code produced by one LTLTRAN statement, save the execution variables, and decode the input statement. After the input line is decoded and the line heading for the next line is sent to the console, execution for some user may resume.

If several users have LTLTRAN programs in execution, the available computer time must be shared. On computers with hardware clocks, a time interval may be allocated for execution. The program being executed will be stopped at the statement transition following a clock produced timing signal. A simpler method, useful on computers with no clock, is to keep a count of the number of program statements executed. When a count limit is reached, the program is stopped and another user's program execution resumed. In the system being discussed, the time required to switch from one user's program to another's is small compared to the program execution time. A simple "round robin" scheduling scheme is adequate for this situation.[3] Each user in execution is allocated an equal time (or statement count) interval, and execution proceeds from one user to another in waiting order.

The computer should send a null character, occasionally, to consoles of users in execution. The null character rattles the console and lets the user know that the computer is still working on the program. This placebo is an important feedback communication to the user. A transmission interval of 1 to 5 seconds is adequate to prevent the deadly lack of action, disturbing to the human user.

Fail safe operation of the system assures that no user's program will be affected by the actions of any other user. The language, LTLTRAN,

---

[3]Scheduling problems and definitions are discussed in reference 13.

eases the system design in this area. LTLTRAN allows only limited access to the computer storage. Each user has direct control of only his own 26 variable storage locations. Thus memory protection is not a required feature of system.

The LTLTRAN error handling, where only a 3 letter mnemonic and current line number need be supplied, allows all errors in any routine to transfer to a common control point. Thus no unforeseen combination of errors can cause a system failure.

The choice of an input LTLTRAN line (statement) as the smallest unit for partial execution or decoding by the system allows independent action by the user for each line. After a line is input or executed, the user may, by use of the intervention features of LTLTRAN, change that line without confusion to the system. The pointers and execution variables, saved for each user, describe a condition between statements, not within a statement.

With limited storage available for user's program, it is not desirable to assign a fraction of the storage to each user. Rather, the available storage should be allocated as needed on a first-come first-served basis. This scheme makes the maximum use of the storage, and allows any user to have a larger program. At any one time it is assumed that a few of the users would have only small programs. When the available storage is all in use, the program building consoles would get an error stop, and would have to wait until some user releases storage by giving the #CLEAR command.

A situation exists where a user may leave the console without giving the #CLEAR command. This display of bad manners may tie up some computer storage, and if the program is in execution, may also be using computer time. To prevent waste from this situation, the system will give an automatic #CLEAR after a suitable time interval from the last character

received from a console. The console bell character should be sent several times at intervals, as a warning, before the automatic clearing takes place. The user can prolong execution by typing any nonprinting character at the bell ring. The time interval before the first bell ring (approximately 5 minutes or an equivalent statement count) should allow for any normal user distraction.

4. IMPLEMENTATION

The previous sections present the design features of an interactive, time share system. This section develops details of the implementation of the system. The implementation is not totally divorced from the design of the system. Many of the design features are included as aids to the user, only because they are inexpensive to implement. Details of the system are shown for an implementation on a PDP-7 computer. Other computers would require different details, but the basic methods should apply.

The computer storage sections are shown schematically in Figure 1.

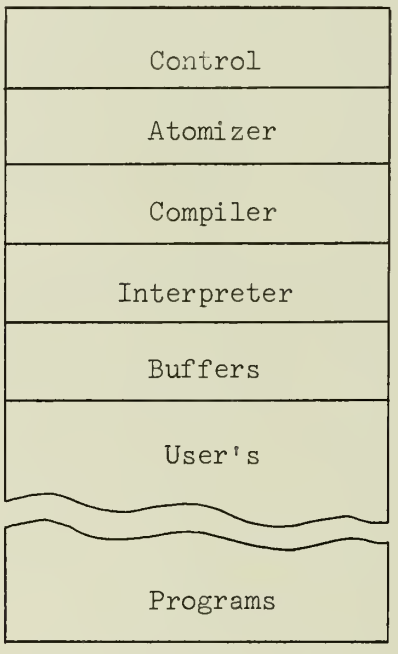


Figure 1

User's Programs

The user's programs are formed as list structures in the space allocated. Each user program is a single, forward linked list. Each list element contains the program information plus a linking address of the next

element in the list. The free space (locations not taken by user's programs) is also linked as a list structure. All elements in all lists are the same lengths. Thus list elements may be attached, inserted, or withdrawn from a list by manipulation of the linking addresses. No consideration need be made of contiguous storage locations, or absolute storage addressing within a list.

The list structuring of the program storage allows the basic features of the system to be easily implemented. LTLTRAN statements are represented by segments of linked elements within the program list. Changing a statement involves replacement of one list segment with another, and has no effect on the storage representation of the rest of the program. During program execution, a LTLTRAN statement transition is recognized by the appearance of a line number (or flag) within a list element. The list segments are linked in line number order, and transfers are executed as a forward search of the list for the label desired.

The list elements are long enough to contain the information relative to the simplest LTLTRAN statement. Complicated input statements map into more than one list element. Figures 2 and 3 present the bit assignments chosen for the PDP-7 implementation. The choice of element size is somewhat arbitrary. For ease of programming, an integral number of storage words per element is required. The number of words per element should be chosen to minimize the number of elements for the types of programs represented. The PDP-7, 3 word element was chosen on the assumption that a larger proportion of the input LTLTRAN statements will be short and not require multiple elements. If the input programs consist of many long, complicated arithmetic statements, a longer element would be justified. A shorter element is not economical, because every input statement would require more than one element, and the ratio of link bits to information

LTLTRAN Element Bit Assignment
(Legend in Figure 3)

	word	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Arithmetic	1	L_i	T_{op}				Link												
	2	S_i	Line										Statement						
	3	O_3					O_2				O_1						A_{op}		
PRINT or READ	1	L_i	T_{op}				Link												
	2	S_i	Line										Statement						
	3	O_3					O_2				O_1						/	N	
GØ TØ or CALL	1	1	T_{op}				Link												
	2	S_i	Line										Statement						
	3	LS	/	/	/	/	/	/	/	Line or Statement									
RETURN or GØ ØN	1	1	T_{op}				Link												
	2	S_i	Line										Statement						
	3	0	/	/	/	/	/	/	/	Line									
END or RUN	1	1	T_{op}				Link												
	2	S_i	Line										Statement						
	3	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/
IF	1	0	T_{op}				Link												
	2	- LS	0 LS	/	Ol L or S							- Line or Statement							
	3	+ LS	/	/	Or L or S							+ Line or Statement							
Constant	1	0	T_{op}				Link												
	2	±	Exponent										1 Fraction						
	3	right Fraction																	

Figure 2

LTLTRAN Element Bit Assignment Legend

Li - Line Number Flag Si - Statement Number Flag
Link - Address of first work of next element
Line - Line Number Statement - Statement Number
N - Number of I/Ø Operands
O_x - Operand Code /// - Unused Bits
LS - Line (0) or Statement (1) Transfer Flag

Type Operation Code (Top)

0 - End of Program
1 - READ
2 - PRINT
3 - GØ TØ
4 - CALL
5 - RETURN
6 - END
7 - IF
10 - Constant
11 - Arithmetic
12 - End of Command
13 - GØ ØN

Arithmetic Operation Code (A_{op})

0 - function	$O_1 \leftarrow O_2 F(O_3)$
1 - assignment	$O_2 \leftarrow O_3$
2 - unary minus	$O_1 \leftarrow -O_3$
3 - addition	$O_1 \leftarrow O_2 + O_3$
4 - subtraction	$O_1 \leftarrow O_2 - O_3$
5 - multiplication	$O_1 \leftarrow O_2 * O_3$
6 - division	$O_1 \leftarrow O_2 / O_3$
7 - exponentiation	$O_1 \leftarrow O_2 ** O_3$

Operand Assignment (O_x)

0 - Constant (in next elements)
1 to 32 - Variables A to Z
33 to 37 - Accumulators 1 to 5

Function Assignment
O_2 Code in O_2F

0 - SIN
1 - CØS
2 - SQRT
3 - ATAN
4 - LØG
5 - EXP
6 - ABS
7 - INT

Figure 3

bits would be too high. The actual bit assignment within an element is a coding detail best left to the programmer.

Constants are carried as part of the program list. Thus no storage buffers need be assigned for them. This will be inefficient for programs that use the same constant many times, where a single storage for the constant would suffice. However, incorporating the constants into the program list simplifies the system programming, allows the user to have as many constants as he wants, and in the normal case, make more efficient use of storage space.

Buffers

Buffers (reserved storage areas) are required: for the 26 variable values allotted to each user, for the incoming or outgoing characters in console communications, and for the pointers used by the system to determine the status of each user. The total length of this buffer storage area is the only design variable directly determined by the maximum number of simultaneous users allowed. It is conceivable that the unused portion of this area could be attached to the free space list when less than the maximum number of users are signed on.

Interpreter

The interpreter is a system program to execute the list elements of the user's LTLTRAN program. This is not a true interpreter in the strictest sense, in that the input string of console characters is not decoded at execution time. The interpreter contains all the system routines to perform floating point arithmetic, change and store variable values, and to search and transfer within the list structure of the LTLTRAN program.

Arithmetic operations are represented in 3-address Polish notation.[1] In this notation, simple arithmetic operations, such as A = B + C or F = CØS (G), are represented by a single element. Code optimization is not a feature of this system, and a floating point store operation is required for each arithmetic task performed. This seeming inefficiency is not too harmful with most small computers, because floating point numbers are not usually held in a hardware register, and thus must be stored in memory in any case. A 5 bit operand code allows up to 5 intermediate result accumulators to be used in evaluating complicated arithmetic expressions. Figure 3 shows the operation and operand codes to be used in the PDP-7 implementation of the system.

Compiler

The compiler is a system program to produce the list elements of the user's LTLTRAN program. This is not a true compiler by strict definition, in that actual machine code is not produced. The use of a system that is part compiling, part interpreting, gives all the flexibility of a true interpretor plus some of the execution speed of a compiled program. The compiler contains routines to construct and link all the types of list elements, and stack manipulating routines to arrange arithmetic elements in proper order for execution.

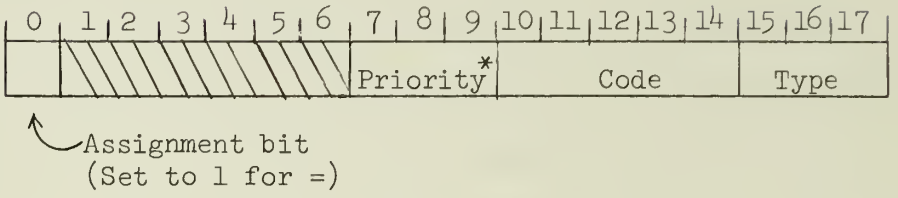
Input to the compiler is an atomized image of the input LTLTRAN statement. The atomizing program of the system maps the console characters into atoms. Each atom is a code indicating the function (type) of the input character (or characters) and the value (token) of the item. The atom bit patterns chosen for the PDP-7 implementation of the compiler is shown in Figure 4.

---

[1] Polish notation is described in reference 14.

LTLTRAN Atom Bit Assignment

(for compiler)



Type

0 - not used
1 - operand
2 - operator
3 - not used
4 - not used
5 - left parentheses
6 - right parentheses
7 - not used

Operator Code	Priority	Symbol
0 - function	6	(F)
1 - assignment	1	=
2 - unary minus	5	~
3 - addition	2	+
4 - subtraction	2	-
5 - multiplication	3	*
6 - division	3	/
7 - exponentiation	4	↑

Operand Code

0 - constant* and dummy operand (d)
1 thru 32 - variable letters A thru Z
33 thru 37 - accumulators ACC_1 thru ACC_5

*Constant temporary storage is addressed by the priority number.

Figure 4

It is the function of the atomizer to encode words, and to convert input numbers and constants. Because of the simplicity of the LTLTRAN language, the atomizer is straight-forward multi-branch programming, proceeding from left to right in one pass through the input character buffer. A dummy operand, (d), is inserted before each unary minus, and a dummy operator, (f), is inserted between each function operand and its argument's left parenthesis. These dummy atoms facilitate the compiling process.

The compiler uses a 3-stack procedure to generate list elements from the atoms of arithmetic expressions. Atoms are withdrawn from the input line buffer, one at a time, in an order corresponding to left to right in the input LTLTRAN line. Operand atoms are pushed into the operand stack. Operators are assigned a hierarchy order as follows:

1 assignment,

2 addition and subtraction,

3 multiplication and division,

4 exponentiation,

5 unary minus,

6 function.

The handling of the operator atom depends on its hierarchy number according to the following:

1. If the operator stack is empty or has a left parenthesis atom on top, the operator atom is pushed into the operator stack regardless of its hierarchy;
2. If rule 1 does not apply and the operator stack has an atom on top of lower hierarchy, the operator atom is pushed into the operator stack;

3. If rules 1 and 2 do not apply, each atom of equal or higher hierarchy is popped from the operator stack until rule 1 or rule 2 applies.

As each operator atom is popped from the operator stack, two operand atoms and one accumulator atom are popped from the operand stack and accumulator stack respectively. These 4 atoms are used to make up the list elements in the order of execution. Additionally, the accumulator atom is pushed into the operand stack. Each accumulator atom popped from the operand stack is used to form the list element, and is pushed back into the accumulator stack.

Left parenthesis atoms are pushed into the operator stack. Right parenthesis atoms cause all operator atoms to be popped from the operator stack, until a canceling left parenthesis atom is popped from the stack. When the input atom buffer is empty, all operator atoms are popped from the operand stack.

The assignment operation is a modification to the last list element formed, and does not form a list element itself. The assignment operation causes the last result accumulator atom to be replaced by the last operand atom in the operand stack.

The arithmetic compiling process is illustrated schematically and by example in the following figures.

Nonarithmetic list elements are generated by standard subroutines within the compiler and atomizer. Commands are processed in exactly the same manner as regular programs except that a program flag is set for future reference, and the command list elements are left in the free space list and not appended to the user's program list.

Control

The most important, and hardest to design, portion of the system implementation, is the system control program. The task of the system control is

LTLTRAN Compiler Schematic

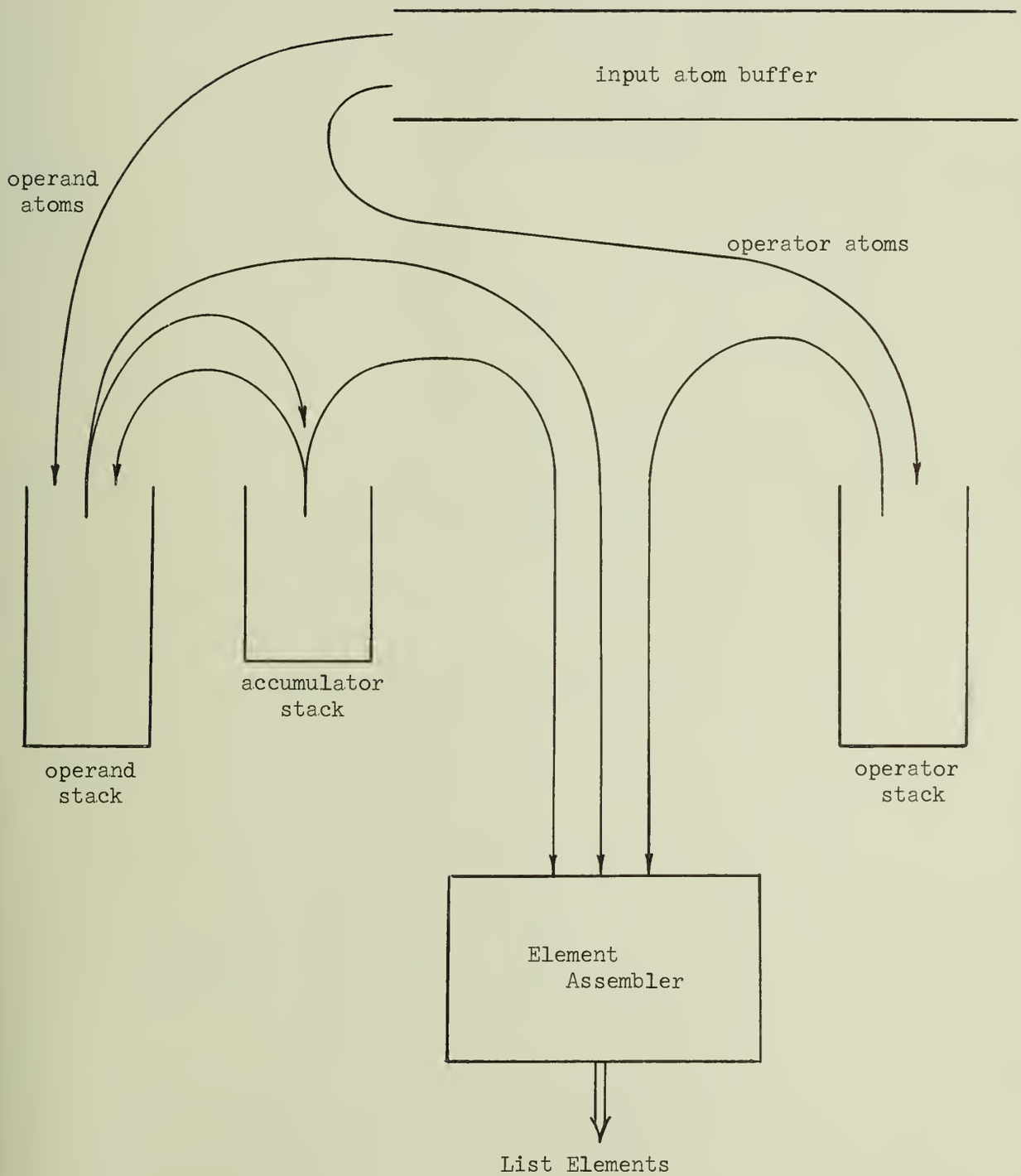


Figure 5

LTLTRAN Compiler Example

LTLTRAN Input Line

A = B * C - H * - SIN (D + E)

Atomized Line

A = B * C - H * ⓓ ~ (SIN) Ⓕ (D + E)

Compiler Output Atoms

*	C	B	ACC_1
+	E	D	ACC_2
Ⓕ	ACC_2	(SIN)	ACC_2
~	ACC_2	ⓓ	ACC_2
*	ACC_2	H	ACC_2
-	ACC_2	ACC_1	ACC_1
=	ACC_1	A	

Order of Output ↓

List Elements Produced

A_{op}	O_3	O_2	O_1
*	C	B	ACC_1
+	E	D	ACC_2
Ⓕ	ACC_2	(SIN)	ACC_2
~	ACC_2	ⓓ	ACC_2
*	ACC_2	H	ACC_2
-	ACC_2	ACC_1	A

Order of Execution ↓

Figure 6

not uniquely determined by a set of inputs and outputs as are other sections of the system. All possible combinations of user actions, for all users, must be allowed for, and the requirements for response time and fail safe operation must be met.

Standard flow charting techniques fail to show interrupt and time dependent program actions. These actions are most important to system control program planning, therefore a new charting scheme was devised to graphically present these program actions and their interrelation. The charting scheme is specifically designed and illustrated for use on the subject system, but it may be extended in scope to cover larger, more complex systems.

For charting purposes all system routines are classified into three types: phantom, tyrant, and interruptable.

The phantom program is specifically identifiable as the program that handles the character-by-character communication with the user's consoles. On computers with hardware interrupt feature, this program is entered at every interrupt. On computers without hardware interrupt, this program is a subroutine called at periodic intervals by all other system routines. For purposes of this report, each entry, by whatever means, to the phantom program will be called an interrupt. A characteristic of the phantom program is that it saves all computer hardware conditions at interrupt, and restores them before returning to exactly the same control point from which interrupt occurred. Thus execution within the phantom program has no effect on the program being executed at the time of interrupt, or vice versa. Program control flags in the phantom program define independent character handling sequences for each console. In effect, each console has its own phantom program running in short spurts at each character interrupt. The phantom program runs in the ID (Interrupts Disallowed) mode so that an interrupt for a character for one console cannot occur during the processing of a character for another console.

A tyrant program is characterized by a programmed change from IA (Interrupts Allowed) mode to ID (Interrupts Disallowed) mode. A tyrant program is used to reset actions in the phantom program. No interrupts can be allowed during running of a tyrant program because of possible ambiguities in control while flags are being set. A tyrant program ends with a change back to IA mode.

Interruptable program runs in the IA (Interrupts Allowed) mode, and includes all program that is not phantom or tyrant. Interruptable program and the phantom program in effect run concurrently with no interaction. Any sequence of the phantom program for a console can be running overlapped with the interruptable program of any other console. A tyrant program runs alone and is not overlapped with any other program.

The essence of the charting method is to classify all system program segments (blocks), and display them in a manner that allows their interrelation to be seen. Each major control path for the system for any <u>one</u> console is shown on a chart. Each chart has the IA (Interrupts Allowed) program blocks shown to the left; ID (Interrupts Disallowed) program blocks shown to the right; all in time order from top to bottom. Tyrant program is indicated on both right and left sides, with the left side block hashed. Every entry and exit from each block is indicated by a labeled arrow. Flag actions are given in the left margin. Timing information, where important, is given in the right margin.

The complete set of charts for the interactive, time share system on a small computer is given in Appendix IV.

The phantom program is controlled by a set of status words, one for each console. Each status word can contain a sequence of action codes to be brought into effect in turn upon receipt of the console end of line character. The action code in effect at a particular instant determines the mode of

the console. The mode of the console is indicated on the planning chart both above and below each set of ID blocks.

A single memory word, called the swap flag, is used to indicate when an executing LTLTRAN program is to be stopped. The swap flag is reset when execution is initiated, and is checked at the execution transition between LTLTRAN statements.

LTLTRAN Console Status Word

Bit	Assignment
0	ignore one interrupt only, and continue
1	error flag
2	not used
3, 4, 5	error code
6	not used
7	not used
8	not used
9, 10, 11	2nd waiting console action
12, 13, 14	1st waiting console action
15, 16, 17	current console action code

Action Code

0	ignore all interrupts
1	receive
2	transmit
3	transmit and echo check
4	intervention, set SWAP upon receipt of printing character
5	receive, set SWAP at SO
6	transmit, set SWAP at SO
7	transmit and echo check, set SWAP at SO

Error Code

0	intervention
1	receiver buffer full
2	echo check failure
3	new console sign-on

Figure 7

5. CONCLUSION

The small computer system developed in this paper permits a general class of arithmetic solutions to be programmed, debugged, and run on the computer in minimum real-time. The human user of the system interacts intimately with the computer to arrive at the problem solution. In the problem solving process, the computer supplies all the routine, preprogrammed labor; the human user supplies input data, and the direction and control intelligence. Since the human user is the controlling factor in the efficiency of the overall process, the system is designed from a human engineering viewpoint, rather than the classical one of computational efficiency. The basic inefficiency of having a slow human be a part of the computing process is compensated by time-sharing several simultaneous users on the computer. Time-sharing does not prevent the computing service to any one user from being degraded by the actions of the other users, however the degradation is spread equitably among all users.

The computer language, LTLTRAN, presented in this paper permits a full range of arithmetic operations on constants and up to 26 variables. The operations may be performed immediately in a "desk calculator" mode, or may be made a part of a computer program for future execution. Program control statements in the language permit unrestricted looping, subroutine calling and nesting, and conditional branching. Additional control conventions allow the human user to make program changes at any point in the program generation, debugging, or running process.

Particular care has been taken in the design of the system to match response time to the requirements of the human user. The major effect of this matching has been to compromise program running performance to gain speed in communications with the user. Another design philosophy permeating the system

is the minimization of console typing. The typing to and from the user is limited to the required information content. However, this philosophy is not carried to the extreme of requiring an extensive artificial vocabulary of abbreviations.

Methods of implementing the system discussed in this report stress the use of list structures for program storage. List structuring permits the flexible control offered the user. Input LTLTRAN lines are mapped into identifiable and maneuverable units in storage. Dynamic allocation of storage allows any user to construct a larger program than would be possible with static, equitable reservation.

Interrupt planning of the system was expedited by the development of a charting system to show program interrelationships. Use of the charting scheme is restricted in this report to the subject system, but the basic details of the scheme should find application in larger, more complex system design. The resulting charts do not replace conventional program flow charts; rather they present information not clearly shown conventionally. Three planning items shown, that are most important are:

1. sequence actions required for the interrupt handling routines,
2. flag actions required to communicate between program segments,
3. overlap opportunities and restrictions.

Evaluation of the system should incorporate examination of features not included. A few of these will be presented here.

Probably the major inadequacy of the system is the inability to handle more than 26 variable values at any one time. This places a direct limit on the scope of problems that may be solved. A user would have to have extraordinary programming ability to use LTLTRAN to solve a set of 5 simultaneous equations (30 input variables, 5 output variables). A similar, related deficiency of the system is the inability to handle variable

arrays, or subscripted variables.

The system does not provide some of the user conveniences common to FØRTRAN. Chief of these is the automatic program loop, or DØ statement. The function of the DØ statement may be programmed in LTLTRAN, but not conveniently. FØRTRAN features of computed GØ TØ, and subroutine local variables may not be much missed in the unsophisticated programming allowed. User defined functions would be a very desirable feature, and may be worth adding later.

The system does not give the user control of the format of the printed output. In particular, messages or headings cannot be printed under program control. Thus LTLTRAN run programs are not self-documenting. Further, it is not possible to obtain a clean print-out of the program. It may be a tedious chore to backtrack a console listing to reconstruct from additions and substitutions a document of the program. A related fault, is the inability to save a program once it is formed. When the #CLEAR command is given, the program is destroyed. To reuse a program, it must be retyped in. Retyping of often used programs may be done automatically from consoles equipped with paper tape readers, but the paper tape must be punched off-line from the computer.

Some of the features of the system presented should be compared with similar features of other systems. The author is most familiar with the systems in the following table, and contrasting details will be drawn from these.

System[1]	Sponsor
CTSS	Project MAC, MIT, Cambridge, Massachusetts
QUIKTRAN	IBM, New York, New York
BASIC	GE, Phoenix, Arizona
SDS/Berkeley	University of California, Berkeley, California

No attempt will be made here to describe these systems, except to note that they all are larger and more extensive than the subject system.

A controversial design point is the handling of the input line. LTLTRAN has the computer supply the input line number as does QUIKTRAN and the CTSS's INPUT program. It is a matter of opinion whether the ease of making changes (the strong point of BASIC and SDS/Berkeley) outweighs the time and trouble savings of the computer supplied labor for normal input. An additional factor in this decision for LTLTRAN, was that the computer supplied ordering of the input would aid the user to produce an orderly, readable listing of the program. This same argument is used to justify the requirement that replacement and insert lines be supplied with a standard heading. It would be very difficult to backtrack on a listing if changed line numbers were buried in the statement text.

Another controversy centers on the end of line character assignment. Some users prefer to use the RETURN key (as in BASIC and SDS/Berkeley) as a "natural" carry over from electric typewriter use. This eliminates a very useful indication of when the computer is ready for a new line of input, as the carriage return provides a clear, audible, and visible portion of the new line heading. The ALT MØDE key is an easy to use end of line key, but is not installed on many teletype consoles. SO, the standard ASCII line terminator, is a combination of the CØNTRØL key and the X key, and is not easy to remember or use (similar to the QUIKTRAN EØB character). LTLTRAN side-

---

[1]The systems are described in references 3, 9, 4, and 11.

steps the controversy by allowing any of the above end of line characters and also the printing character semicolon, ; .

None of the comparison systems have the LTLTRAN double labeling scheme.[2] CTSS and QUIKTRAN allows a class of commands and program control statements to refer to one: the line number or the statement number, but not to either interchangeably. BASIC allows only a line number, with the attendent difficulties of forward reference.

Of the dedicated (single language) systems, QUIKTRAN and BASIC, only QUIKTRAN is interactive, and that only for programs with no subroutines. BASIC programs may be interrupted during execution, but no access to program-generated variables may be gained, and execution cannot be resumed.

From a human engineering viewpoint, the QUIKTRAN and SDS/Berkeley systems offer experience upon which response time decisions can be made. SDS/Berkeley does not offer "immediate" response to each character, but rather delays printing until .2+ seconds after each key is pressed. This has the effect of slowing the typing of even experienced users. QUIKTRAN accepts characters as fast as they may be typed, but does not process input lines rapidly. The user must wait 2 to 10+ seconds between input lines. This delay encourages the user to try to fit complicated expressions on one line with the attendent waste typing in case of error. QUIKTRAN (a rather slow, fully interpretive system) does nothing to the console during program execution. This leaves the user with the nagging question, "Has the system died, or is it just running slow?"

Future work on the interactive, time share system, beyond the scope of this report, should include:

1. implementation of the design on a real computer,

---

[2] A few of the QUIKTRAN commands may refer to either label, but the scheme is not consistently applied, and only statement numbers may be used in program.

2. investigation of additional features that may be desirable,

3. extension of the design to larger, more powerful computers.

Study of the system under actual operation should show up the most desirable additional features, as well as prove the usefullness of this design.

6. BIBLIOGRAPHY

[1] Eric Burgess, et al, On-Line Computing Systems, American Data Processing, Inc., Detroit, Michigan, 1965.

[2] R. S. Burington, Handbook of Mathematical Tables and Formulas, McGraw-Hill Book Company, New York, N. Y., 1965.

[3] F. J. Corbato, et al, The Compatible Time-Sharing System: A Programmer's Guide, 1st ed., M.I.T. Press, Cambridge, Massachusetts, 1963.

[4] Dartmouth, BASIC, Dartmouth College Computation Center, Hanover, New Hampshire, 1964.

[5] DEC, Reference Manual PDP-7, F-75, Digital Equipment Corporation, Maynard, Massachusetts, 1965.

[6] DEC, 630 Data Communication System, F-03 (630A), Digital Equipment Corporation, Maynard, Massachusetts, 1964.

[7] W. P. Heising, "History and Summary of FORTRAN Standardization Development for the ASA," Comm. ACM, 7, October, 1964.

[8] IBM, FORTRAN II, General Information Manual, GENL-25, IBM Corporation, New York, N. Y., 1961.

[9] IBM, IBM 7040/7044 QUIKTRAN User's Guide, IBM Corporation, New York, N. Y., 1965.

[10] IBT, Bell System Services, ASCII Code, 33 and 35 Teletypewriters, Illinois Bell Telephone Company, Chicago, Illinois, 1964.

[11] W. W. Lichtenberger and M. W. Pirtle, "A Facility for Experimentation in Man-Machine Interaction," AFIPS Conference Proceedings, Vol. 27, Part 1, Spartan Books, Washington, D. C., 1965.

[12] P. Naur, et al, "Revised Report on the Algorithmic Language ALGOL 60," Comm. ACM, 6, January, 1963.

[13] A. L. Scherr, An Analysis of Time-Shared Computer Systems, MAC-TR-18 (Thesis), MIT, Cambridge, Massachusetts, 1965.

[14] Peter Wegner, et al, Introduction to System Programming, Academic Press, New York, N. Y., 1964.

[15] W. E. Woodson, Human Engineering Guide for Equipment Designers, University of California Press, Berkeley, California, 1956.

APPENDIX I

SYNTAX OF LTLTRAN

The syntactic definition of LTLTRAN is given in Backus[1] notation. The metalanguage brackets, < >, delimit the objects of the nonterminal vocabulary.

The metalanguage separator, |, may be read as "or".

<letter>	A|B|C|D|E|F|G|H|I|J|K|L|M|N|Ø|P|Q|R|S|T|U|V|W|X|Y|Z
<digit>	1|2|3|4|5|6|7|8|9|0
<line number>	<digit> <digit> <digit>
<statement number>	<digit>|<digit> <digit>
<label>	<line number>|<statement number>
<unsigned integer>	<digit>|<unsigned integer> <digit>
<integer>	<unsigned integer>|
	+ <unsigned integer>|
	- <unsigned integer>
<number>	<integer>|<integer> .|
	<integer> . <unsigned integer>|
	. <unsigned integer>|
	+ . <unsigned integer>|
	- . <unsigned integer>
<constant>	<number>|<number| E <integer>
<variable list>	<letter>|<variable list> , <letter>
<command word>	CLEAR|GØ ØN
<statement word>	RETURN|RUN|END
<function>	SIN|SINF|CØS|COSF|ATAN|ATANF|LØG|LØGF|
	EXP|EXPF|SQRT|SQRTF|INT|INTF

[1]Backus notation here is a subset of the notation described in reference 12.

<primary> <constant>|<letter>

(<arithmetic>)|<function> (<arithmetic>)

<factor> <primary>|<factor>**<primary>|<factor>↑<primary>

<term> <factor>|<term>*<factor>|<term>/<factor>

<arithmetic> <term>| + <term>| - <term>|

<arithmetic> + <term>|

<arithmetic> - <term>

<assignment> <letter> = <arithmetic>

<I/Ø statement> READ <variable list>|PRINT <variable list>

<GØ TØ statement> GØ TØ <label>

<IF statement> IF (<arithmetic>) <label>, <label>, <label>

<CALL statement> CALL <label>

<statement> <assignment>|<I/Ø statement>|

<GØ TØ statement>|<IF statement>|

<CALL statement>|<statement word>

<command> # <line number>|<line number>|

<statement>|

<command word>|<command word>

<line> <line number> SO|

<line number> <statement> SO|

<line number> <statement number> <statement> SO|

<line number> <command> SO

APPENDIX II

SEMANTICS OF LTLTRAN

The semantics of LTLTRAN follows that of FØRTRAN.[1] The following sections describe and give examples of the features of LTLTRAN.

Variables and Constants

LTLTRAN allows 26 variables to be assigned values. Each variable is referenced by a single letter of the alphabet, A through Z. Each letter refers to a unique storage location for a floating point number. Thus, in LTLTRAN, as in FØRTRAN, a letter variable is a label for a storage space; not in an algebraic sense, a label for a numeric value. For example: A = A + 1.5 is a valid, meaningful LTLTRAN statement.

LTLTRAN stores and handles only floating point numbers (fraction and exponent). The usual FØRTRAN fixed point (integer value) conventions do not apply. All input numbers may be in any standard FØRTRAN format, with or without decimal point. The letter E in a constant indicates a following exponent for a multiplying power of ten.

Algebraic Operations

The arithmetic operations to be performed by LTLTRAN are specified by assignment statements. Assignment statements have the form: letter = arithmetic expression. The letter to the left of the assignment symbol, =, specifies the storage location for the result value of the expression to the right. The arithmetic expression may be any legal combination of letter variables, constants, functions, and operators.

The LTLTRAN operations include addition, subtraction, unary minus,

[1] References 8 and 14.

multiplication, division, and exponentiation. LTLTRAN exponentiation allows only integer powers, and gives the correct result sign. An attempt to calculate a fractional exponentiation will result in an error stop.

Parentheses pairs, ( ), may be used to define complicated expressions. Excess (redundant) parentheses pairs will be ignored. The standard FØRTRAN conventions of hierarchy of operations is followed. Expressions enclosed in parentheses pairs are evaluated first, from the innermost pair outward. Functions are evaluated second. Within parentheses pairs and otherwise, arithmetic operations are evaluated in the following order:

1. unary minus,
2. exponentiation,
3. multiplication and division,
4. addition and subtraction,
5. assignment.

Multiple operations of the same type are evaluated from left to right.

LTLTRAN operations are indicated by the following teletype symbols:

Operation	Symbol
assignment	=
addition	+
subtraction	-
multiplication	*
division	/
exponentiation	** or ↑
unary minus	-
parentheses pairs	( )

Examples of LTLTRAN algebraic operations:

```
A = -B + C
F = F - I* (C - E↑H)
```

```
G = (X - 3) * ((J * K - 9.3) - 8/C + L ** 2)

C = 3.4567E-5

Y = (((Y ** 2) ** -3) +2)
```

Functions

LTLTRAN computes some of the standard FØRTRAN functions. Functions are indicated by the function code followed by an argument expression within parentheses pairs. A function may be used as a subexpression within a more complicated expression.

The standard function codes vary from one version of FØRTRAN to another. LTLTRAN will accept as equivalent, various codes for the same function. The LTLTRAN function codes and function definitions are given in the table below.

Function Code	
SIN SINF	Trigonometric sine of the argument. Argument in radians.
CØS CØSF	Trigonometric cosine of the argument. Argument in radians.
ATAN ATANF	Trigonometric arc tangent (radians) of the argument. Argument assumed to be in first two quadrants.
SQRT SQRTF	Square root of argument. (Error stop on negative argument)
LØG LØGF	Natural logarithm of argument. (Error stop on negative argument)
EXP EXPF	Natural exponent (antilogarithm of argument)
ABS ABSF	Absolute value of argument.
INT INTF	Integer portion of argument.

Examples of LTLTRAN function expressions:

```
A = SQRT (B)

X = C - LØG (Y - Z)

T = SIN (A) / CØSF (A)

Z = -ABS (INT (G * Y + EXP (C + D))) + 3.4
```

Control Operations

To facilitate program control, each LTLTRAN statement may have one or two labels. One label, supplied by the computer, is a 3 digit line number. The user can supply an additional 1 or 2 digit statement number. The line number label is always 3 digits preceded by a decimal point. The statement number label is never 3 digits, and does not have a decimal point. Thus the two labels are readily distinguishable.

The statement number may be supplied or not, in any order, by the user. The statement number is compatible with the standard FØRTRAN system, and facilitates reference to subsequent statements.

The line number is supplied, in numerical order, by the computer as a part of the input process. The line number facilitates reference to previously input statements without statement numbers. In the absence of control statements, the LTLTRAN statements are executed in order of ascending line number.

All LTLTRAN control statements may refer to either the line number label or the statement number label, at the option of the user. The following sections describe the LTLTRAN control operations: GØ TØ, IF, CALL, RETURN, END, and RUN.

The LTLTRAN unconditional transfer is the GØ TØ statement. The words GØ TØ are followed by a statement label. Upon execution of the GØ TØ statement, control is transferred to the statement with the label specified.

Examples: .015 GØ TØ 3

.155 6 GØ TØ .013

.572 22 GØ TØ 67

Conditional branching if effected by an IF statement; the word IF, followed by an arithmetic expression enclosed in a parentheses pair, followed by three statement labels separated by commas. Control will transfer to the statement labeled with the first, second, or third specified label if the result value of the arithmetic expression is negative, zero, or positive respectively.

Examples: .020 IF (A - B) 3, 4, 15

.355 28 IF (ATAN (C)) .035, 61, .040

.460 3 IF (C * F - (G - ABS (X + I))) 5, .012, .012

LTLTRAN subroutines are defined by the label (line number or statement number) of the first statement in the routine. The last statement of a subroutine is the word RETURN. The transfer to a LTLTRAN subroutine is by a CALL statement; the word CALL, followed by the label of the first statement of the subroutine. Upon execution of a subroutine RETURN statement, control transfers to the statement following the CALL statement.

Variables referenced in a subroutine are global to the calling program. No distinction is made between the actions of a subroutine statement and any other statement, except for the RETURN. Transfer to a subroutine by any method other than a CALL statement will cause an error stop upon execution of the RETURN statement. A LTLTRAN subroutine may include calls to other subroutines, but such calls must not be recursive, i.e. a subroutine may not call itself.

The END or RUN statement may be used to start execution of a LTLTRAN program. Upon input of the word END or RUN, control is transferred to the first statement of the program (the statement with the lowest line number). If execution of an END or RUN statement is attempted, an end of program stop will occur. The use of END for this function is compatible with FØRTRAN.

The LTLTRAN language features presented so far are used to build and run a program. These features resemble those of the standard FØRTRAN, and are not concerned with the interactive, man-machine nature of LTLTRAN. The following section presents the semantics of those features of LTLTRAN concerned with console handling and real-time control of the program.

Input/Output Operations

Values for letter variables may be input from the console at the execution of a READ statement. A READ statement is the word READ followed by a list of the letter variables separated by commas.

Examples: .105 READ A, B

.200 3 READ X

.255 READ L, M, N, X, A, C, Z

Upon execution of the READ statement, the computer will type the line number of the statement, followed by up to three of the letters in the list. The user will then supply the new values for the letter variables specified. The input values, separated by commas, may be input in any standard format. A plus sign, +, with no number may be input to indicate no change to the value of the letter variable. For READ statements with variable lists longer than three letters, more than one input sequence will be executed. Examples (user input underlined. SO is end of line character.):

.105 A,B: 2.5, 3.697 SO

.200 X: 10.3E - 7 SO

.255 L, M, N: 5, 7.3, - .0007 SO

.255 X, A, C: +, +, 9.65E3 SO

.255 Z: 8.65 SO

Values of letter variables will be output to the console at the execution of a PRINT statement. A PRINT statement is the word PRINT, followed by

a list of the letter variables separated by commas.

Example: .305 8 PRINT X, C

.390 PRINT A, B, N, L, Z

.585 26 PRINT M

Upon execution of the PRINT statement, the computer will type the line number of the statement, followed by up to three of the variables values in the list. The output sequence will be repeated for PRINT statements with more than three letters in the list.

Examples:

.305 X=+.103000E-05 C=+.965000E+04

.390 A=+.250000E+01 B=+.369700E+01 N=-.700000E-03

.390 L=+.500000E+01 Z=+.865000E+01

.585 M=+.730000E+01

Input of program statements is by a stereotyped input line. The LTLTRAN input line always begins with the computer output of: carriage return, line feed, line number, space, and X-ØN.[2] No user input is required, or allowed until this standard heading is supplied.

The LTLTRAN line number is incremented, by the computer, in steps of .005 to allow insertions. This interval allows up to 199 normal input lines, with an allowance for inserting up to 4 additional lines between each pair of normal lines.

A replacement or insert line may be obtained by input of a statement consisting of the line number desired. The computer will then take in the one line requested in the standard format.

---

[2] X-ØN is to start the console paper tape reader on consoles so equipped. Each line on input paper tapes must have a X-ØFF character punched two characters before the SO character ending the line. The teletype console must have the standard automatic reading features.

Example, insert a line between lines .015 and .020 (user input underlined. SO is end of line character.):

.065 .017 SO

.017 3 A = B + 2.5 SO

.065

A line may be deleted by input of a null statement, consisting of only the end of line character, SO.

Immediate Execution

LTLTRAN statements may be executed normally in a program, as described previously. In addition, an individual statement may be immediately executed, as a command. Any of the LTLTRAN statements, except CALL and RETURN, may be preceded by a number sign, #, to cause immediate execution. Assignment statements input as a command, allow LTLTRAN to operate as a "desk calculator". Transfer statements used as commands initiate execution of the LTLTRAN program at the statement indicated.

Commands are not saved as part of the LTLTRAN program. The line number of a command is not incremented for the next input. Statements that are logically immediately executed, are not affected by a preceding number sign, #, i.e. RUN and #RUN are semantically equivalent.

Examples: .205 # A = B + C

.615 # PRINT X,Y

.660 # H = 3.9

.755 # IF (A - G) 5, .015, 13

Execution Interruption

The user may regain control of a LTLTRAN program during execution, by typing any printing character. The computer will respond with a standard

error stop, and an input line heading. Execution may be continued from the point of interruption by input of the command GØ ØN (or #GØ ØN). Any number of commands or program modifications may be made between interruption and resumption of execution.

Console Sign-on and Sign-off

The user may initiate action for input to LTLTRAN by typing any printing character. The computer will respond with the input line heading for line .005.

The command CLEAR (or #CLEAR) will destroy the user's program, release any computer storage used, and zero the values of the letter variables for the console. The CLEAR command may be input by the user to sign-off, or may be issued automatically by the computer to release an inactive console.

Console Special Characters

LTLTRAN uses a terminal vocabulary of standard teletype ASCII[3] characters. The use of letters, digits, and some operators have been given. Some other teletype characters are used for special functions.

The teletype character SO (control X) is used to indicate the end of every input line. The characters semicolon, ;, ALT MØDE, and RETURN are equivalent to SO and may also be used to indicate the end of line. Redundant end of line characters will be ignored by LTLTRAN.

The left pointing arrow character, ←, is the LTLTRAN backspace. Backspace may be used to correct typing errors. Each backspace character will remove one preceding printing character.

---

[3]The teletype version of the ASCII code is given in reference 10.

For example, the line:

.105 A = C + D ← ← * G SO has the same meaning as:

.105 A = C * G SO

The per cent character, %, will cause the containing line to be totally ignored by LTLTRAN. This character may be used to indicate a comment line, or to delete action on a line with errors. The character may appear in any position in the line.

Nonprinting characters, except SO, ALT MØDE, and RETURN, are ignored by LTLTRAN. Thus SPACE, TAB, and LINE FEED may be used freely to format the printed input.

Printing characters that have no defined LTLTRAN meaning are illegal, and will cause an error comment.

Error Printout

All errors and nonstandard program actions produce a standard error message to the user. The message consists of a 3 letter mnemonic, indicating the source of the error, followed by the line number label of the last line acted on by the computer. This error indicator is followed by a line heading for a new input line or command from the user.

Example (the printout produced by a floating point overflow upon execution of statement .020):

ØVF .020

.165

APPENDIX III

LTLTRAN EXAMPLE

The following example is a simulated session at a console. The teletype printing to the left of the page is the console input/output, triple spaced for readability. The user input is underlined. The problem being worked on is the evaluation of the following definite integral:

$$v = \int_{x=1}^{x=u} \frac{(ax)^n}{\log ax} \, dx$$

Consultation of integral tables produces the following:

$$v = F(u) - F(1)$$

$$F(x) = a^n \int \frac{x^n}{\log ax} \, dx = \frac{1}{a} \left[\log|\log ax| + (n+1) \log ax + \frac{(n+1)^2 (\log ax)^2}{2 \cdot 2!} + \frac{(n+1)^3 (\log ax)^3}{3 \cdot 3!} + \ldots \right]$$

$$F(x) = \frac{1}{a} \left[\log|\log ax| + \sum_{k=1}^{\infty} \frac{(n+1)^k (\log ax)^k}{k \cdot k!} \right]$$

Two subroutines are used:

1. a subroutine to calculate F (x),

and 2. a subroutine to accumulate the summation to a given accuracy.

Several errors are shown to illustrate the LTLTRAN error statement.

Variables

V - result value of integral (v)

F - F (x)

U - upper limit of integration (u)

L - lower limit of integration (1)

A - constant (a)

N - exponent (n)

G - log ax

S - intermediate value of sum

K - summation index

Ø - temporary storage for sum

Q - $(n+1)^{k-1}$

T - $(k-1)!$

P - $(\log ax)^{k-1}$

```
$                                  Any printing character will
                                   connect a console to the system.

.005  READ A,N                     Input statement to
                                   get the values for
                                   variables A and N.

.010 1 READ L,U                    Statement to get values for L and U.
                                   Statement number is 1, Line number is .010 .

.015  X=U                          Set variable X to upper limit value.

.020  CALL 3                       Subroutine call to calculate F(X) .
                                   Subroutine starts at statement number 3 .

.025  V=F                          Temporarily save F(U)
                                   in storage V .

.030  X=L                          Set X to lower limit.

.035  CALL 3                       Calculate F(L) .

.040  V=V-F                        Calculate V=F(U)-F(L) .

.045  PRINT V                      Output statement to
                                   print the value of V .

.050  GO TO 1                      Transfer statement to repeat
                                   calculations for new values of U and L .

.055 % SUBROUTINE TO CALCULATE F(X)   Comment ignored by system
                                      because of % character.

.055 3 G=LOG(A*X)                  Start of subroutine (statement 3).
                                   Uses LOG function to calculate G .

.060  S=LOG(ABSF(G))

.065  CALL 2                       Subroutine call to calculate summation.

.070  F=S/A

.075  RETURN                       End of subroutine 3.
```

```
.080 % SUBROUTINE TO CALCULATE SUM
.080  K=1%
.080 2 K=1
.085  P=1
.090  Q=L1←←1
.095  0=S
.100  K=K+1
.105  P=P*G
.110  Q=Q*(N+1)
.115  T=K*T
.120  S=S+Q*P/(K*T)
.125  IF (ABS(S/0-1)-.00001) 5,5,.095
.130 5 RETURN
.135  RUN
.005 A,N:  1. , 1.
.010 L,U:  0. , 1.
     0VF .120
```

Comment

Forgot statement number.
Delete line with % character.

Start of subroutine 2 .

Correct error with backspace.
LTLTRAN will read: Q=1 .

Save old value of S for comparison.

Conditional transfer
comparing summations.

End of subroutine 2 .

Command to start execution.

Test values input
for A and N .

Error in execution of line .120 .
Overflow during execution.

.140 # Z=K*T

Command to calculate a value for Z .
character causes immediate execution.

.140 # PRINT Z

.140 Z=+.000000E+00

.140 # PRINT K,T

.140 K=+.100000E+01 T=+.000000E+00

.140 # .092

Command to insert a line at .092 .

.092 T=1

Input of insert line.

.140 # GO TO .010

Restart execution at line .010 .

.010 L,U: 1. , 5.

.045 V=+.316223E+02

.010 L,U: .1 , 1. ?

Printing character ? typed to stop execution.

INT .025

Execution was stopped at line .025 .

.140 # PRINT S,Q,P,K,T

.140 S=-.291122E+02 Q=+.200000E+01 P=-.22737E-12

.140 K=+.200000E+01 T=+.200000E+01

.140 GO ON

Resume execution from point of interruption.

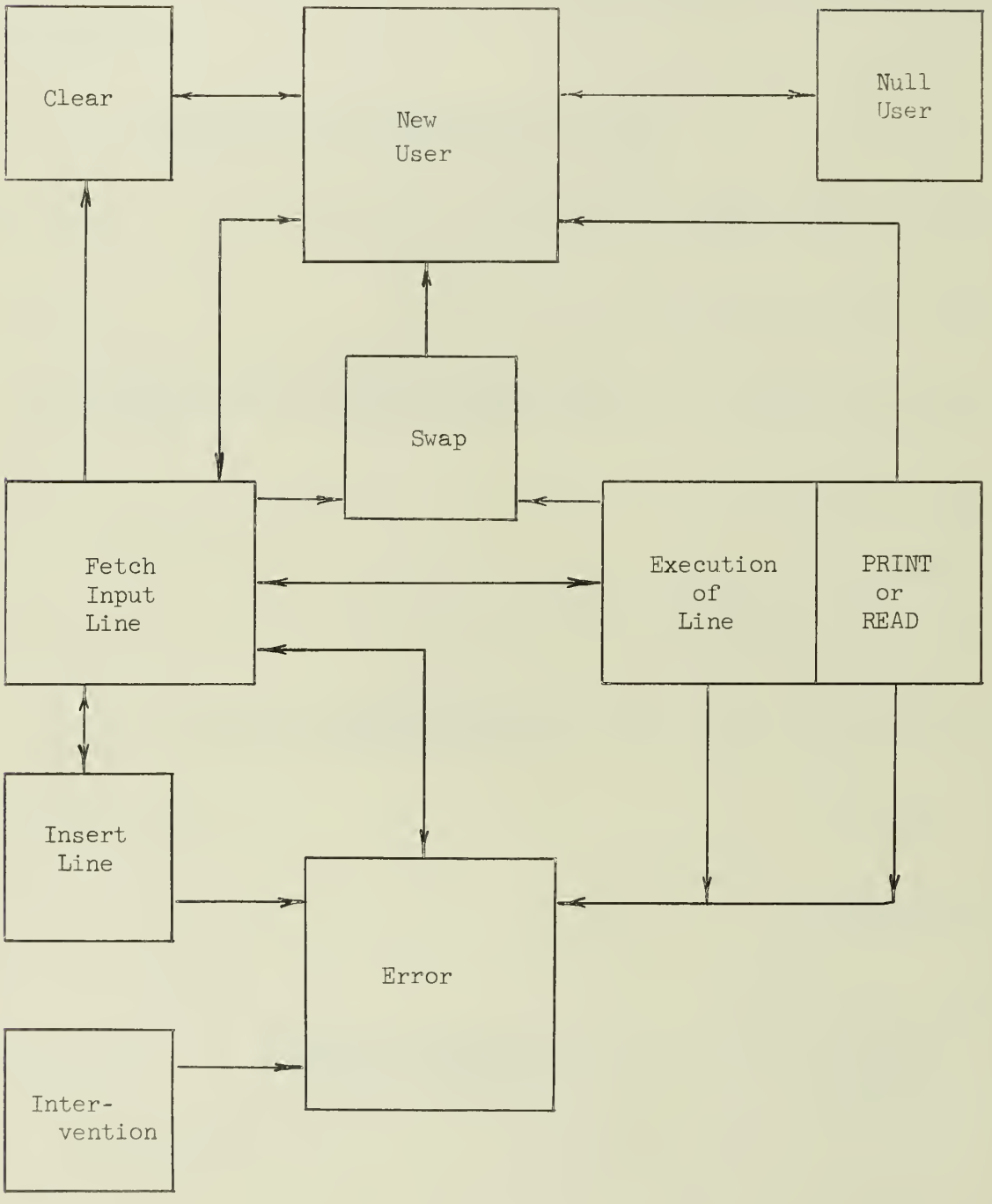
.045 V=-294036E+02

```
.010 L,U: ?                    Illegal input character to stop execution.

     ILC .010

.140 # A=SQRTF(1.43*(.97+5.9512))

.140 # N=3

.140 # GO TO 1                 Restart execution at statement 1 .

.010 L,U: .356 , 6.25

.045     V=+.370051E+03

.010 L,U: + , 2.89             Plus sign convention to keep L=.356 .

.045     V=+.327237E+02

.010 L,U: .985 , +

.045     V=+.305000E+02

.010 L,U: + , + ?              Interrupt execution with ? .

     INT .065

.140 # PRINT A

.140     A=+.314600E+01

.140 # CLEAR                   Console sign-off.
```

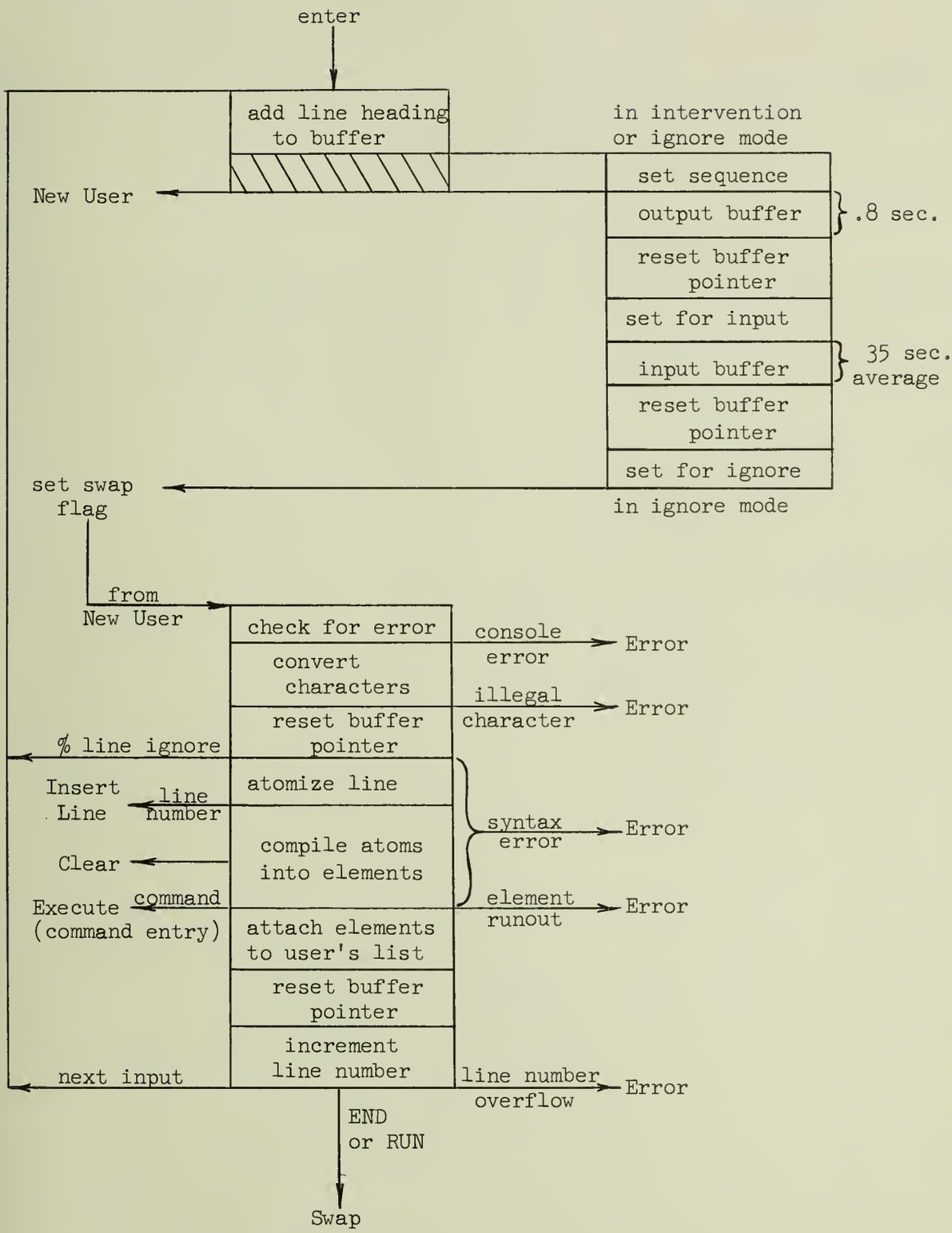
APPENDIX IV

SYSTEM DESIGN CHARTS

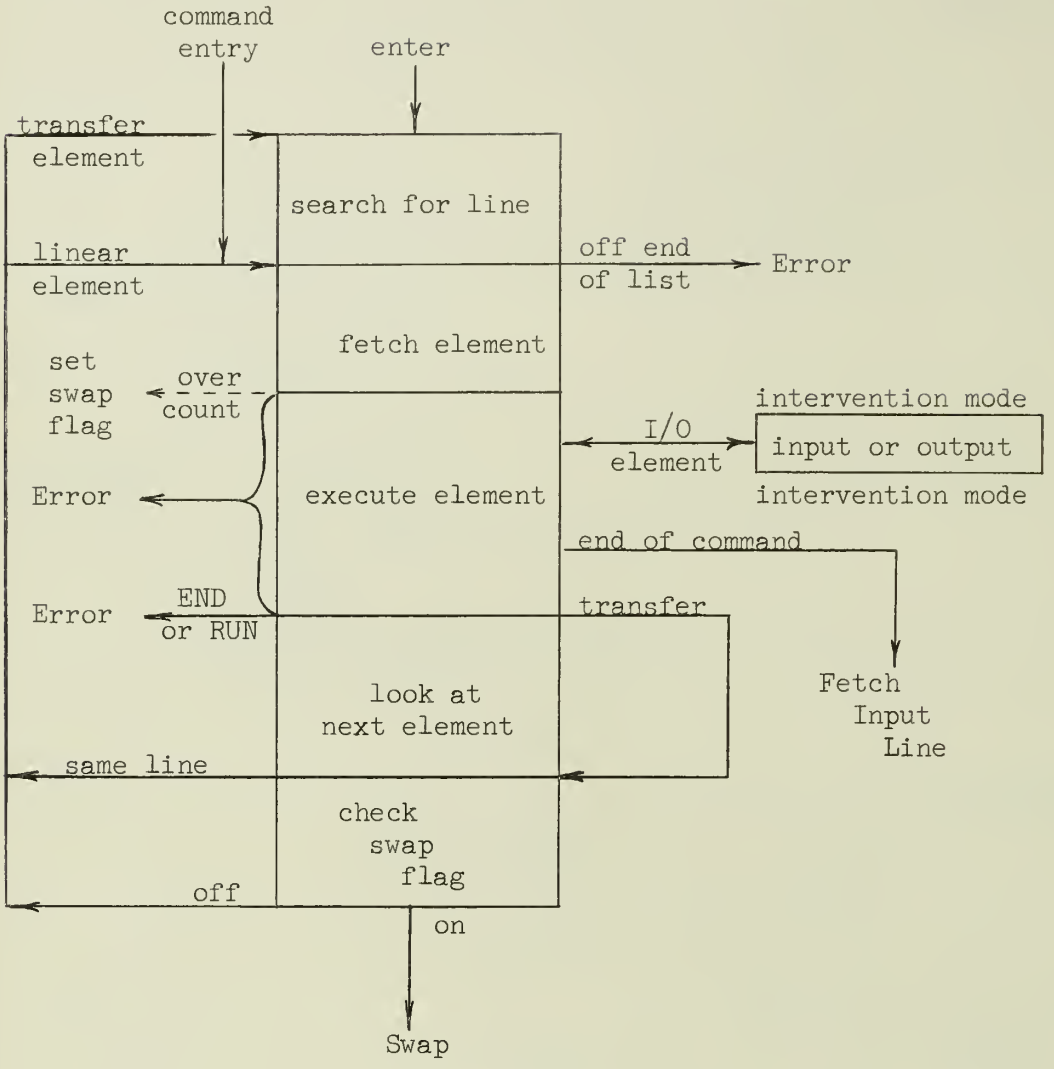
SYSTEM CONTROL SCHEMATIC



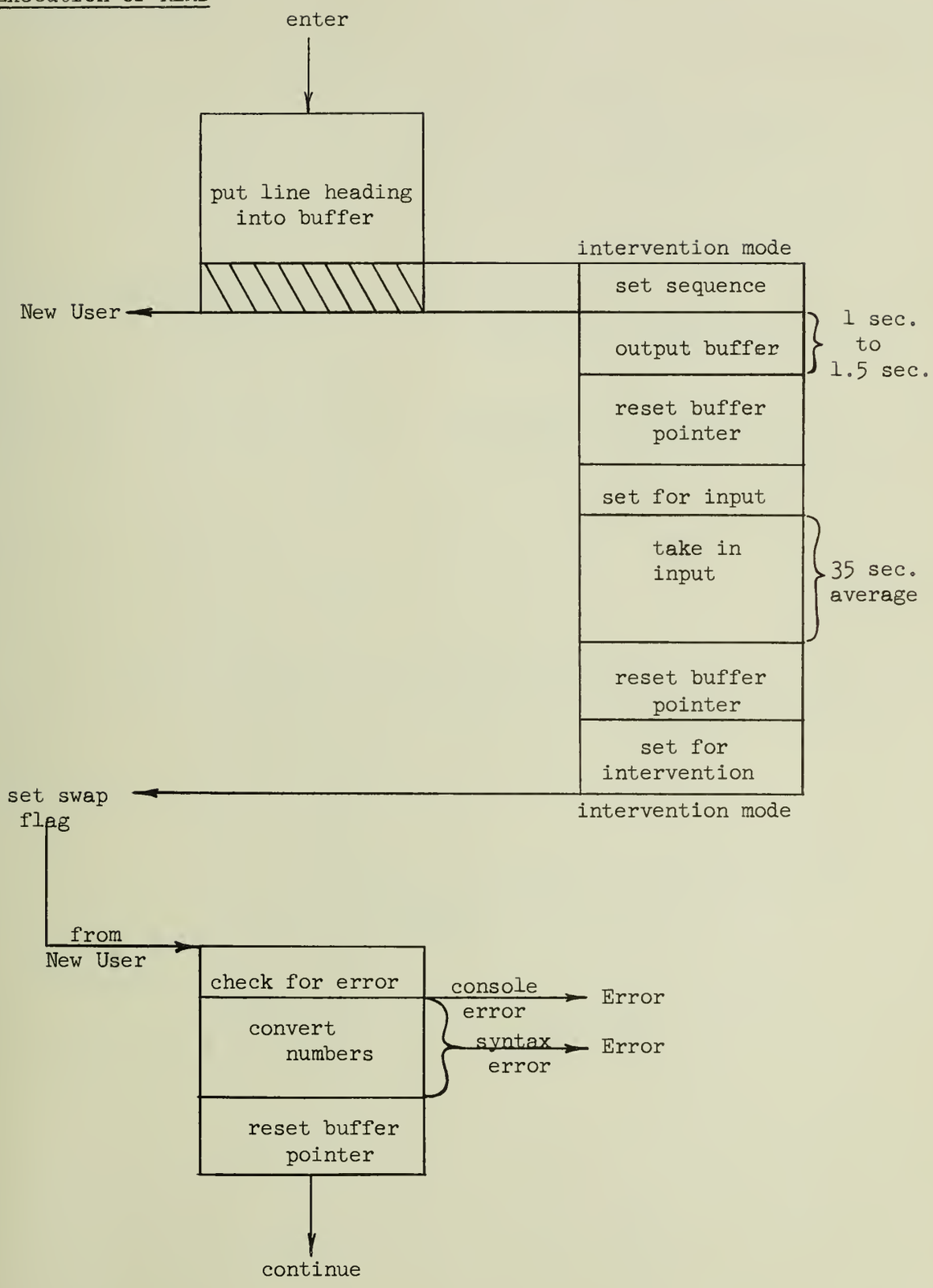
Fetch Input Line



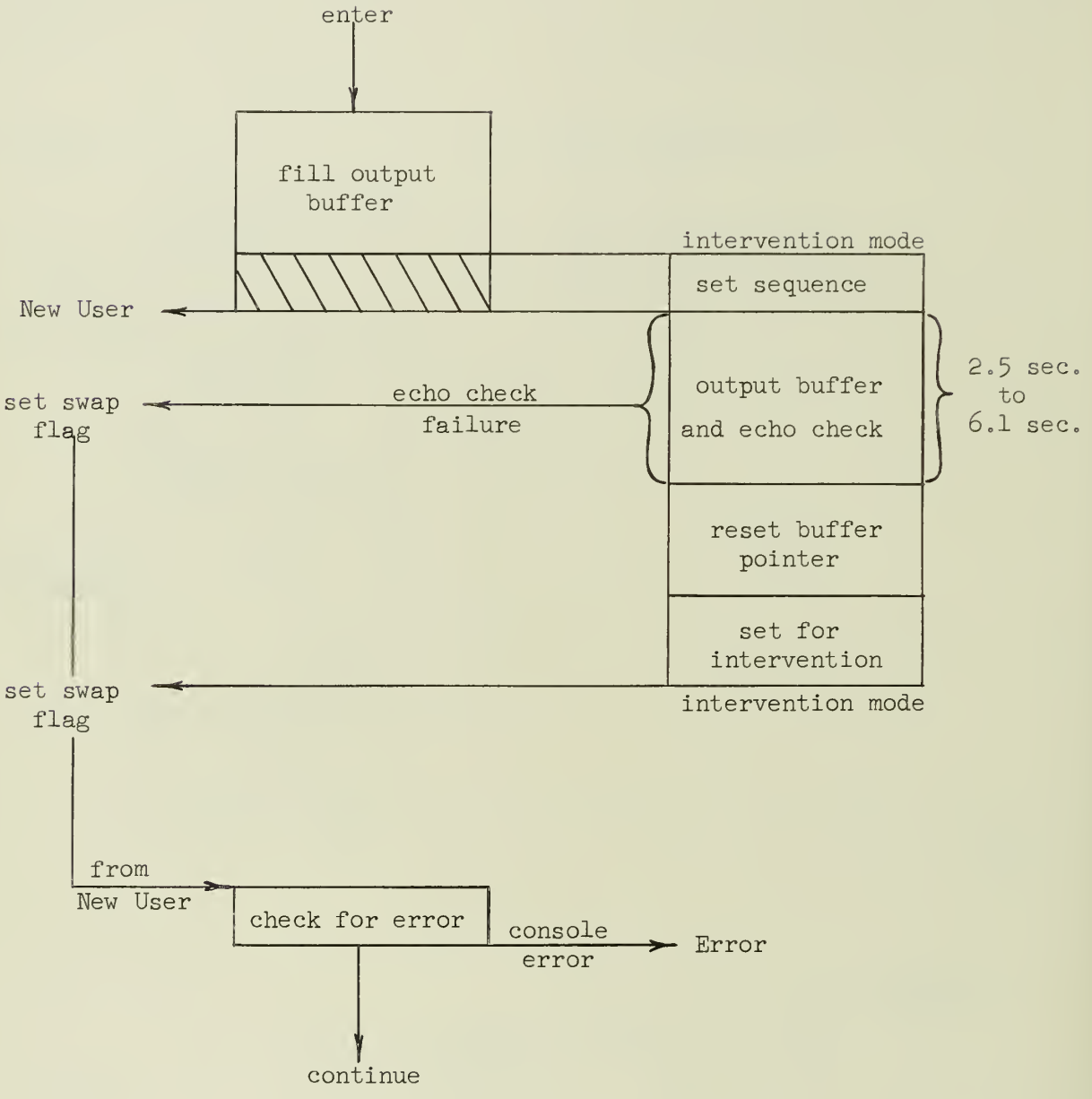
Execution of Line



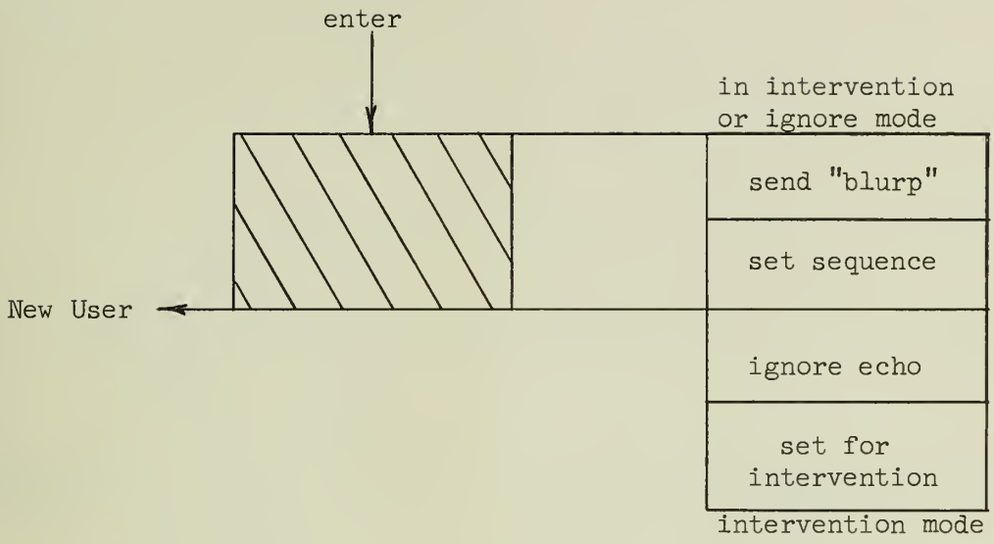
Execution of READ



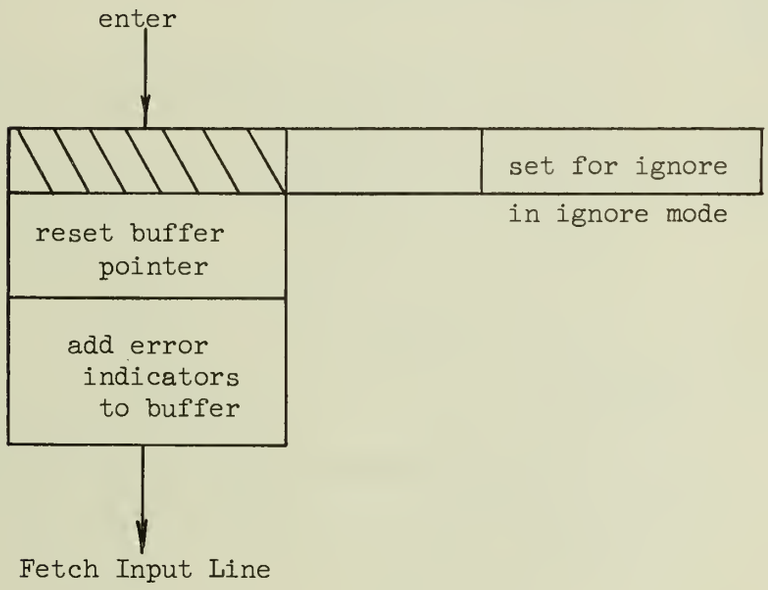
Execution of PRINT

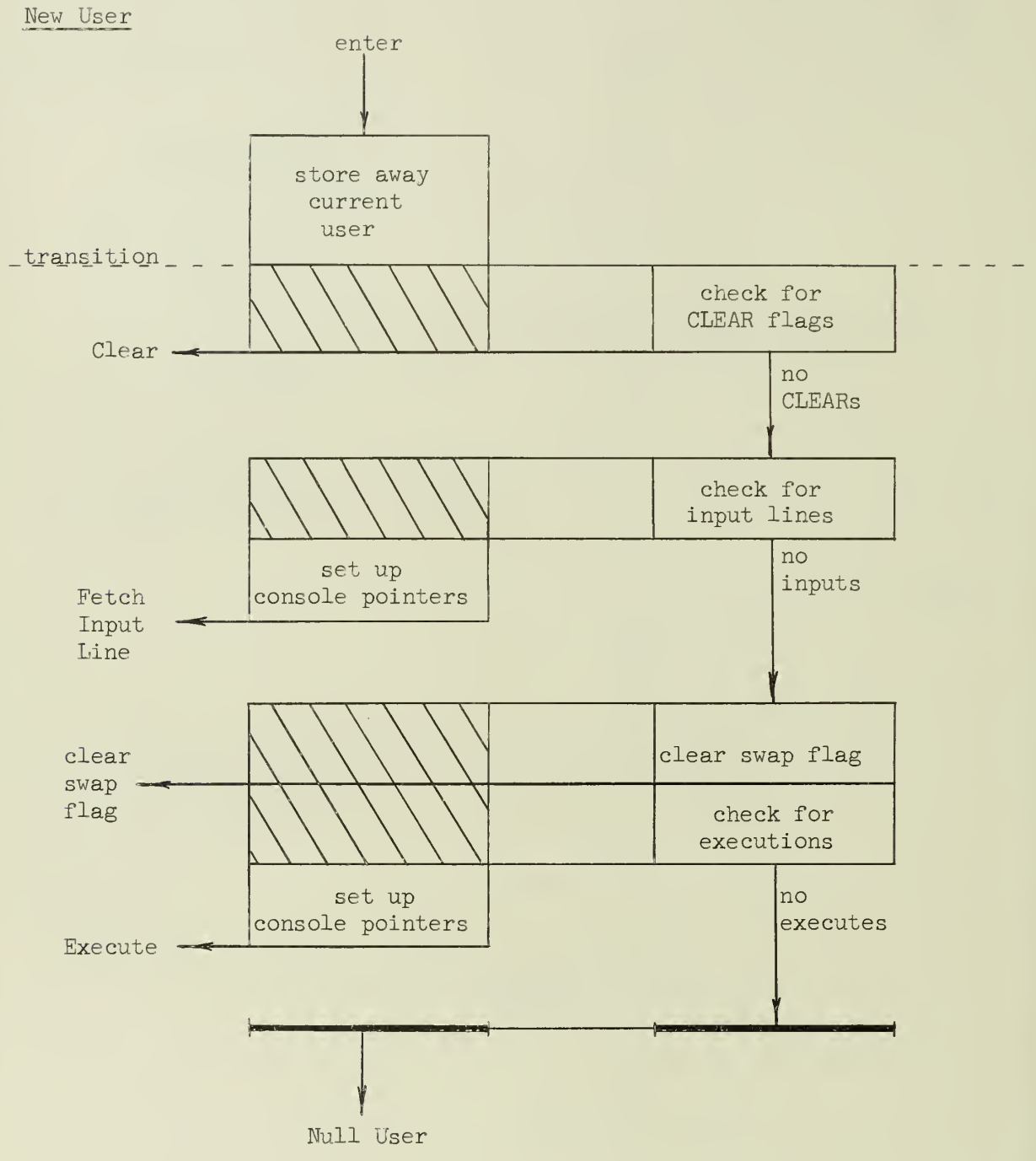


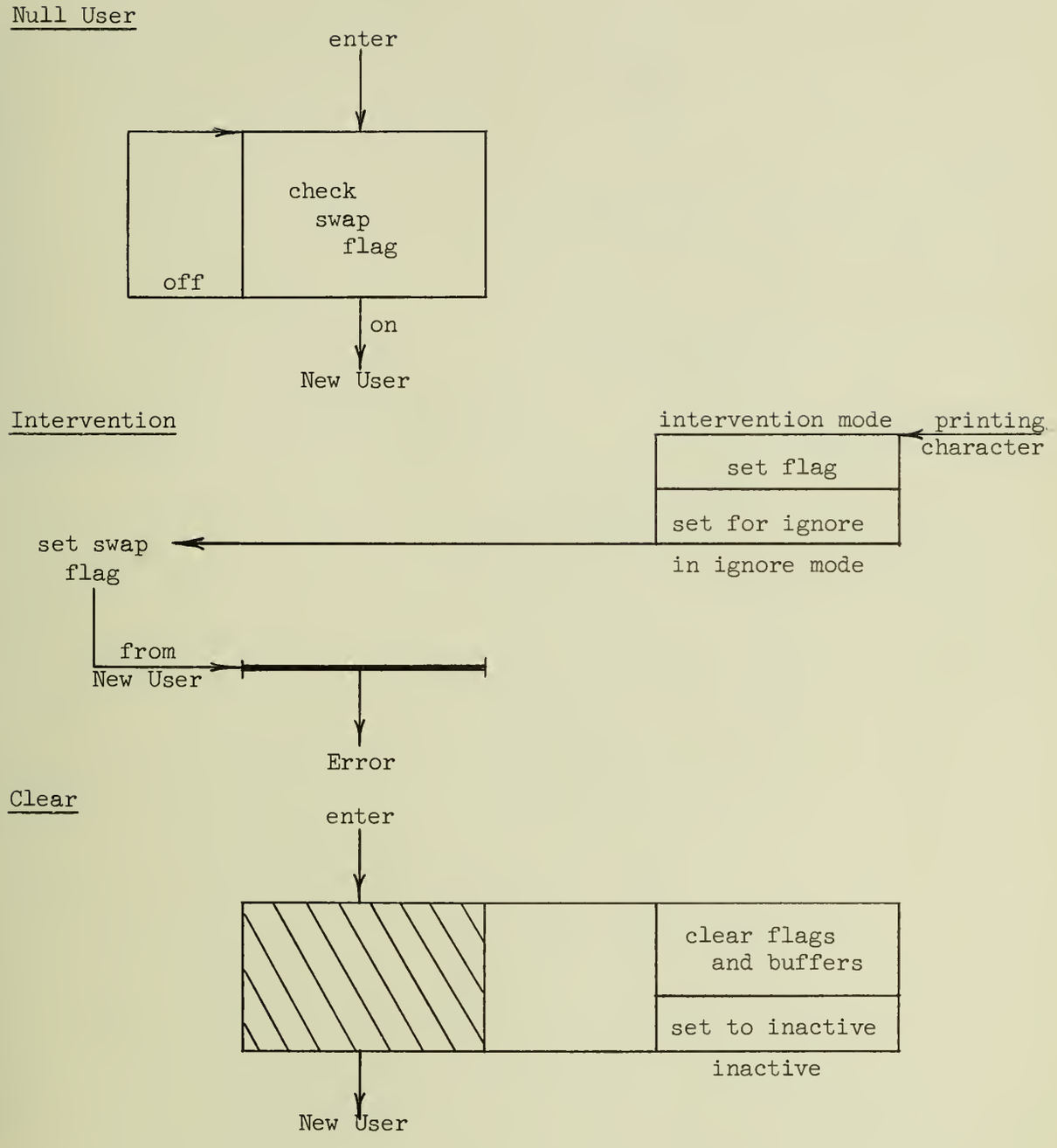
Swap


enter
in intervention
or ignore mode
send "blurp"
set sequence
New User
ignore echo
set for
intervention
intervention mode

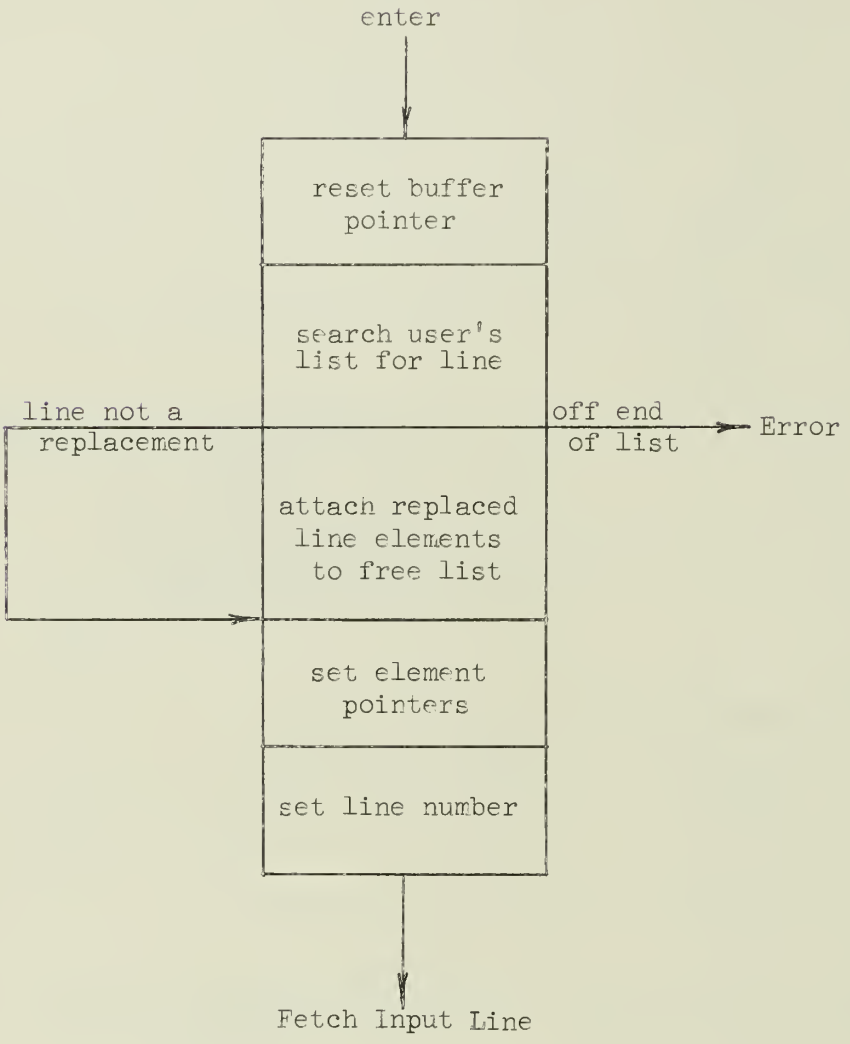
Error


enter
set for ignore
in ignore mode
reset buffer
pointer
add error
indicators
to buffer
Fetch Input Line


New User
enter
store away current user
transition
check for CLEAR flags
Clear
no CLEARs
check for input lines
set up console pointers
Fetch Input Line
no inputs
clear swap flag
clear swap flag
check for executions
set up console pointers
no executes
Execute
Null User


Null User
enter
check
swap
flag
off
on
New User
Intervention
intervention mode
printing
character
set flag
set for ignore
in ignore mode
set swap
flag
from
New User
Error
Clear
enter
clear flags
and buffers
set to inactive
inactive
New User

Insert Line


enter
reset buffer pointer
search user's list for line
line not a replacement
off end of list
Error
attach replaced line elements to free list
set element pointers
set line number
Fetch Input Line